NCC2-5120 Summary of Research
The Simulation of Read-time Scalable Coherent Interface

Qiang Li[2], Terry Grant[1], Radhika S. Grover[2]

1 NASA Ames Research Center
2 Department of Computer Engineering, Santa Clara University

1 Introduction

Scalable Coherent Interface (SCI, IEEE/ANSI Std 1596-1992) [SCI1, SCI2] is a high performance interconnect for shared memory multiprocessor systems. In this project we investigate an SCI Real Time Protocols [RTSCI1] using Directed Flow Control Symbols. We studied the issues of efficient generation of control symbols, and created a simulation model of the protocol on a ring-based SCI system. This report presents the results of the study.

The project has been implemented using SES/Workbench. The details that follow encompass aspects of both SCI and Flow Control Protocols, as well as the effect of realistic client/server processing delay. The report is organized as follows. Section 2 provides a description of the simulation model. Section 3 describes the protocol implementation details. The next three sections of the report elaborate on the workload, results and conclusions. Appended to the report is a description of the tool, SES/Workbench, used in our simulation, and internal details of our implementation of the protocol.

2 Protocol Implementation Details

2.1 SCI Model

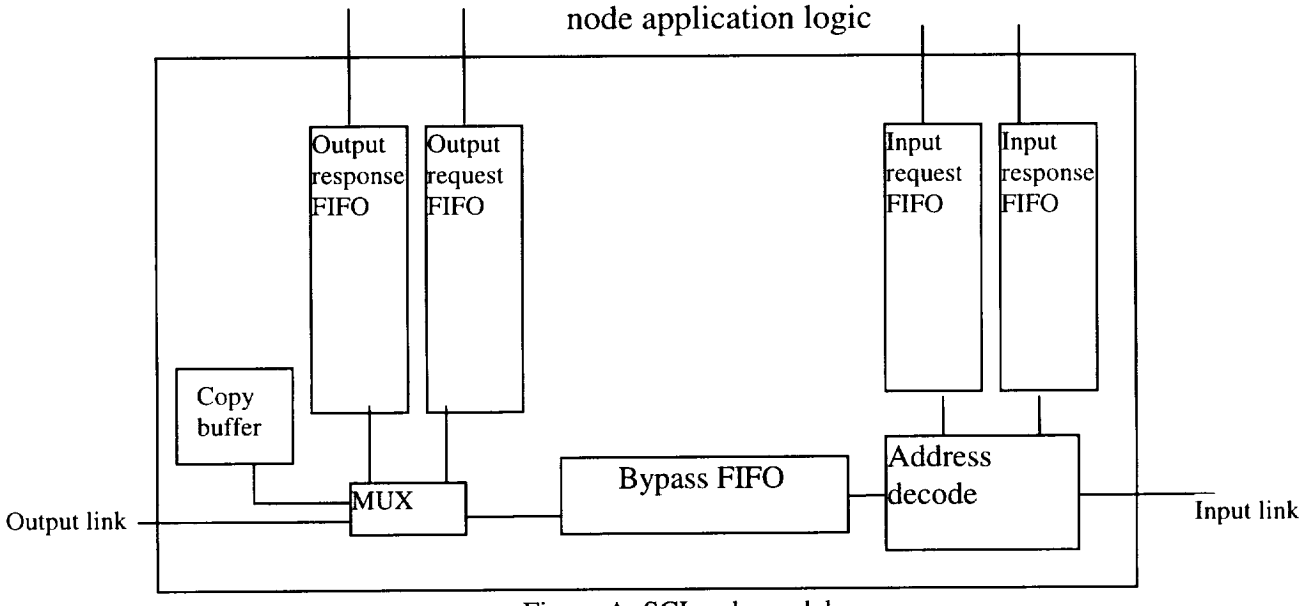

Figure A: SCI node model

SCI provides bus services with point-to-point links operating at gigabyte/second speed. While traditional buses allow only a single transfer at a time SCI allows multiple concurrent independent transfers. Therefore, SCI is more scalable that the traditional buses.

There are no topological standards for SCI system configurations. Small systems can be implemented as rings. Larger systems can be implemented with bridges connecting two or more ringlets (rings with a small number of nodes). In our study, we simulate a system of a single ring with eight nodes.

The SCI node contains prioritized input and output queues and a bypass FIFO as shown in Figure A. A node can transmit a new packet only when its bypass FIFO is empty. This is specified by SCI to prevent deadlock. Whenever a *request* or *response* packet is transmitted it is saved in the *copy buffer* until an *echo* packet is returned from the addressed target node. The size of the copy buffer is implementation dependent. We assume the size to be 4 in our study. If a *done echo* is returned the copy of the sent packet is discarded otherwise it is resent. *Idle* symbols are always sent between packets.

2.2 Real Time Extension

The input queues are modified to handle various priority levels. Each input queue has the specified number of priority levels and each priority level can hold a maximum specified number of packets (four in this simulation). Whenever a packet of a certain priority arrives for the input queue, if the corresponding priority level is filled, the packet is placed in the next highest /lowest priority level available. If there is no space left, the packet is *busied* (a busy echo is sent back to the sender and the packet is discarded).

The packets in the higher priority level are processed before those in the lower priority level. Within a priority level FIFO ordering is used. Once a packet has been removed from a queue it is processed fully before getting the next packet from the queue (i.e. a newly arrived higher-priority packet cannot pre-empt the current packet being processed). The input queues have HIGH and LOW watermark levels as will be discussed in details later.

Control symbols are used to transmit information that restricts *(stop control symbol)* or restarts (*start control symbol*) flow of packets to a node (destined to the node) or through a node (passing through the node)..

Figure B shows the additional registers in an SCI node which implement the Flow Control Protocol [RTSCI1]. Each SCI node has two *DIQS* registers (Destination Input Queue Status) and one *Restricted Traffic register*. The DIQS registers are updated using control symbols to inform all the nodes on the ring of those nodes which have their input queues filled for a certain priority level. Similarly, the Restricted Traffic register in each node is updated by a control symbol to reflect those nodes that have their bypass FIFO blocked. The information in these registers is used to send_new packets based on priority.

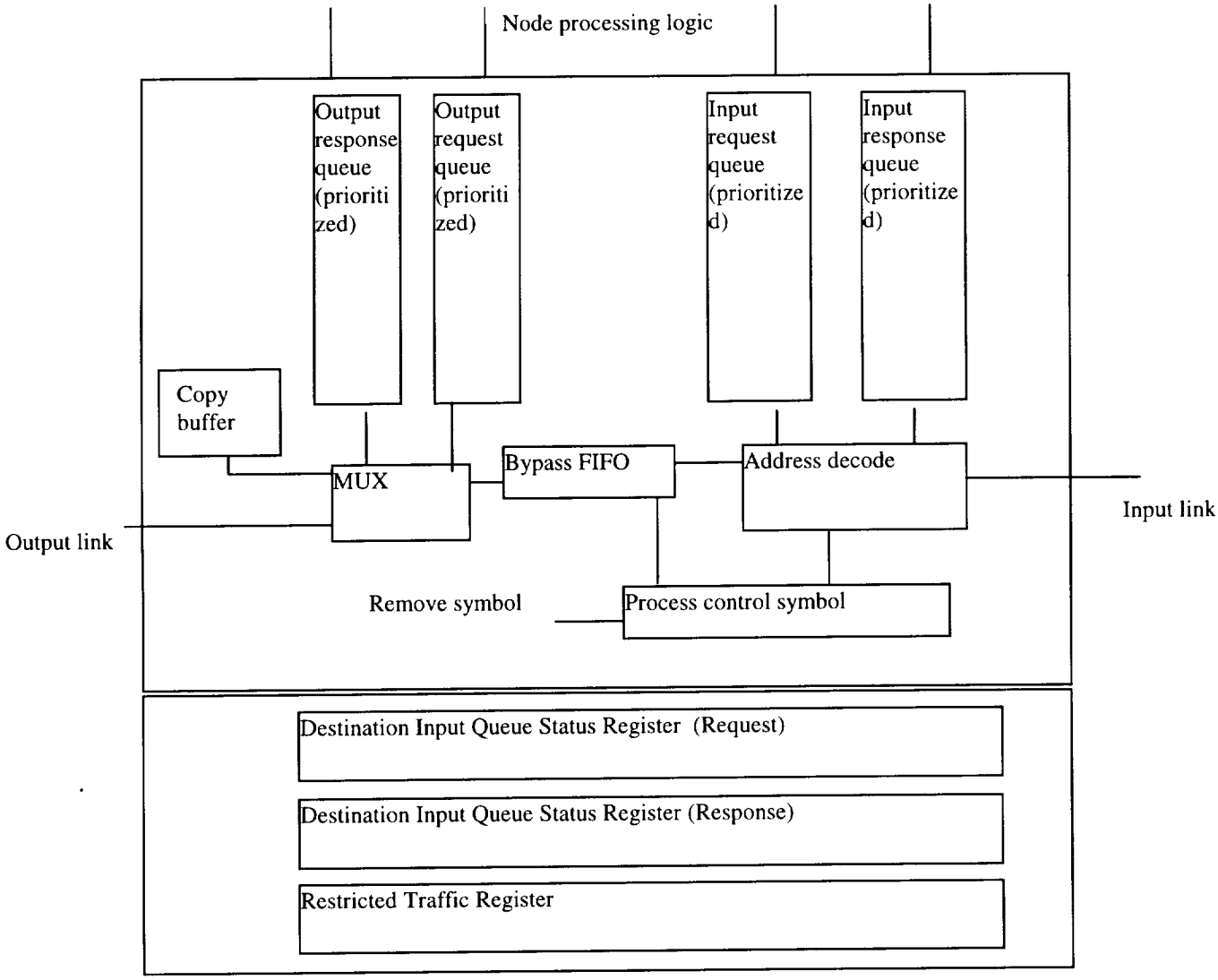

Figure B: SCI Node with real time extensions

2.2.1 Control symbol generation for input queues

Whenever a node's input queue is full (or near full), the node sends a control symbol around the ring to restrict the transmission of packets by other nodes to it until a second control symbol that allows transmissions to the node to resume is received. This is detailed below.

When the HIGH watermark is reached and the lower priority levels are filled to capacity, the node status is set to *blocked* and a *stop control symbol* for input queues is sent out for that priority level. This control symbol updates the DIQS registers of all nodes. A *start control symbo*l for the input queue is generated if the node is blocked and when the LOW

watermark is reached for the priority level specified in the *Busied Packet register*. A start control symbol is also generated for a blocked node when the input queue is cleared completely. This updates the DIQS registers of nodes around the ring telling them that the input queue of the node, which sent out this control symbol, is no longer blocked. The registers to hold status for input request and response queues are used since they can reduce the number of unnecessary control symbols circulating through the ring. This is achieved by setting the status bit to *blocked* when a stop control symbol is sent for a certain priority. If the HIGH watermark is reached again for the same priority level another control is not generated since the status bit is set.

Also, in the current implementation, the lowest priority of packets that were busied is stored in the Busied Packet Register (as opposed to the highest priority used in [RTSCI1]). Since a range of priorities may be critical, the lowest priority busied within that range could also be used instead.

RTSCI specifies that an *agent* node is used to activate the "start" control symbol to maintain fairness and is rotated around the ring.

2.2.2 Generating a control symbol for bypass queues

To allow a node to transmit a higher priority packet than that in the bypass FIFO the *stop bypass control symbol* is generated and sent to all other nodes on the ring. This is required since a node injects new packets only when its bypass FIFO is empty. Every time a request or response packet arrives in the bypass FIFO the output queues is checked to see if there is a higher priority packet in them. If there is such a packet then send out a *stop bypass control symbol* to disallow all priorities lower than the priority in the output queue through the bypass so that the output queue is able to transmit that packet. Whenever a packet of higher priority (than that being blocked currently) arrives in the bypass a check is made to see if another control symbol for bypass needs to be generated. This is necessary for the protocol to be scalable to larger number of priorities. The *start bypass control symbol* for the bypass is sent out by the node when the output queue is cleared of ALL packets with priority higher than that specified in the Restricted Traffic register for that node. This follows since the node has transmitted the packet(s) from the output queue and the bypass FIFO does not need to be blocked any longer. No agent node is required here (as opposed to control symbol generation when input queue is full) and already activated control symbols are sent.

The output queues transmit a packet only if its priority is greater than the priority restrictions in the DIQS register of destination node and in the Restricted Traffic register of any node between source and destination. Otherwise the packet is held aside and re-evaluated every time a control symbol arrives at the node. These packets are re-evaluated for transmission in a prioritized manner and before any other packets that may have arrived in the output response and request queues.

2.3 Simulation Model

In our implementation of the protocol the number of nodes and priorities (limited only by existing memory and time for the simulation to run) are configurable parameters. The protocol has been implemented at symbol level. A random packet generator creates packets with Poisson inter-arrival times and places them in a non-prioritized queue of finite length. The priorities and destination nodes are assigned to the packets using random uniform distribution. The SCI and Flow Control protocols have been implemented using SES /Workbench.

3 Workload

The system was modeled as a ring with 8 nodes. In one case the nodes are sending randomly to all other nodes in the ring. In the second case a "hot spot" was created with the nodes sending 80% of packets to one node and the remaining 20% randomly distributed. The bandwidth on the link between any two nodes is set to 1000 MB/sec. Request packets are 16 bytes and responses are 80 bytes. Control symbols are independent packets, 4 bytes each, generated when required by a node. The simulation kept a track of the maximum number of control symbols in the bypass at any time.

The simulation was run for a two-priority system for varying inter-arrival rates. We also varied the delay between arrival of requests and sending response (it corresponds to the parameter "delay" specified in the graphs). This delay represents the time taken by the processor or memory at the receiving end to service a request (it is assumed to be zero for responses). The latencies were measured for the high priority and low priority packets for various system throughputs. The latencies presented below, however, are the network latencies and do not include this service delay time. The average latency, throughput and the number of control symbols generated were measured. Statistics were collected after an initial warm-up time during which was tested if the system was converging. We deemed it sufficient to collect data for 7000- 8000 packets for a system of size as ours. The maximum latencies of high and low priority packets were also measured

4 Results

4.1 The graphs showing average and maximum latencies of high and low priorities vs. the offered load:

We analyzed how the network behaved under different request packet service times for the two models. We also turned off the generation of the bypass control symbols (allowing control symbols to be generated for the input queues) and measured system performance.

It can be seen, in general, that when the bypass control symbols are generated (Figures 1 through 6) the system favors high priority packets as the offered load reaches the point near saturation. When the system is lightly loaded, the average latencies are almost the same.

As expected, for the same request processing time (labeled *delay* in the graphs) the

average latencies in the model with a hot spot were longer than those in the random model. Since the majority of packets (80%) are addressed to a single node in the model with a hot spot, the input queues of that node fill up quickly, causing more packets to be busied and hence the higher latencies. It is also interesting to see how the request processing delay at the input request queue can affect the latencies. In the random model there was almost no difference in the average latencies for the 0ns (ideal case) and 300ns processing delay (Figures 1 and 3). However in the model with a hot spot the average latencies for both priorities are higher with a 300ns processing delay (Figure 4) and the system is saturated at about 35% load offered in the ideal case (Figures 2). Again this can be attributed to an increase in the busied packets resulting from filled input queues of the hot spot node being inundated with packets from other nodes around the ring. The performance is further deteriorated with processing delays of 3000ns in both random and hot spot models (see Figures 5 and 6) since packets have a larger average waiting time in the nearly or completely filled input request queue.

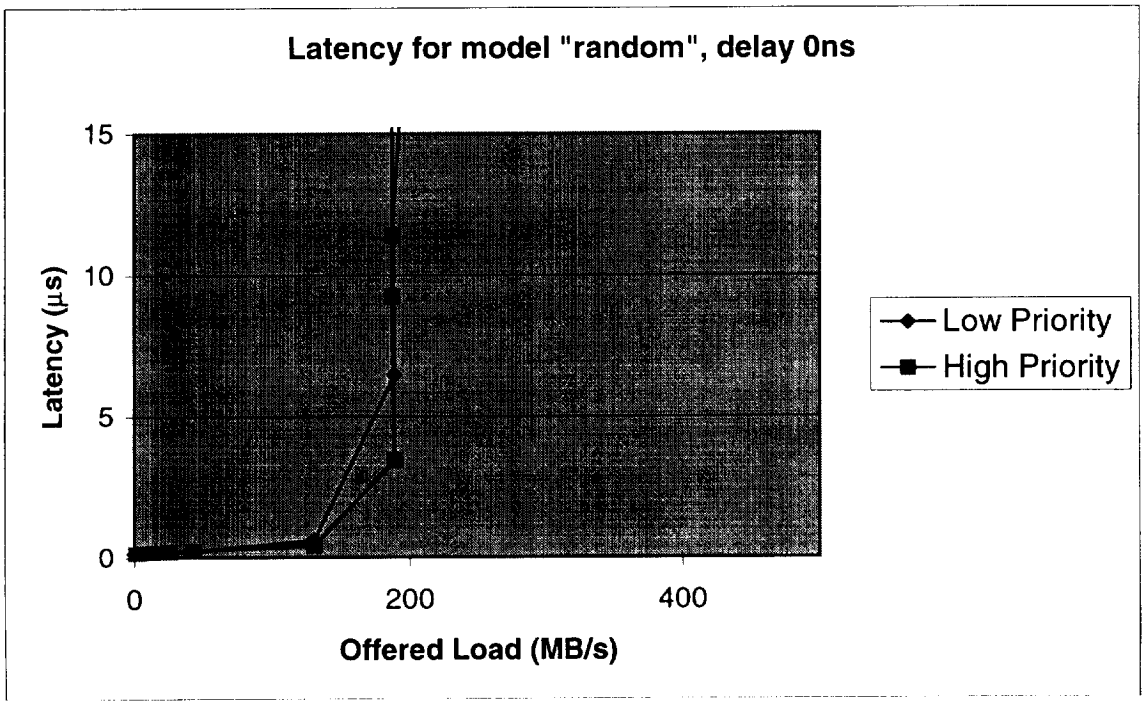

Figure 1: Latencies of low and high priority packets vs. the offered load for a system with delay 0ns

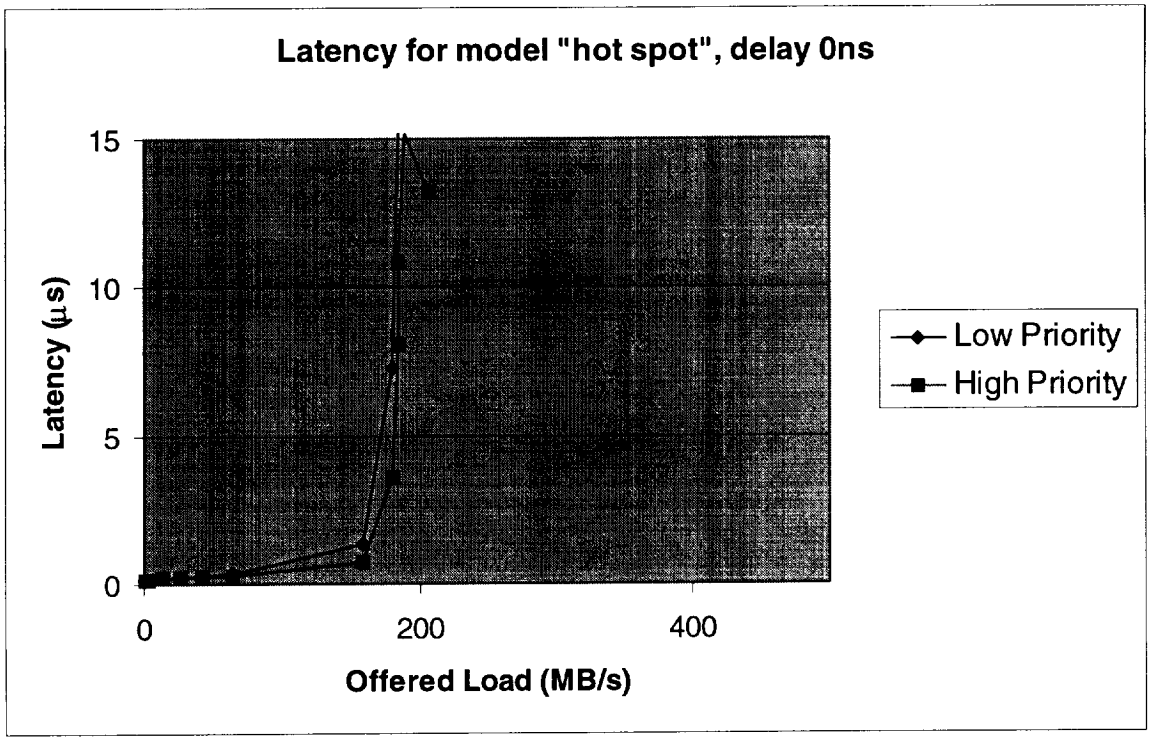

Figure 2: Latencies of low and high priority packets vs. the offered load for a system with hot spot and delay 0ns

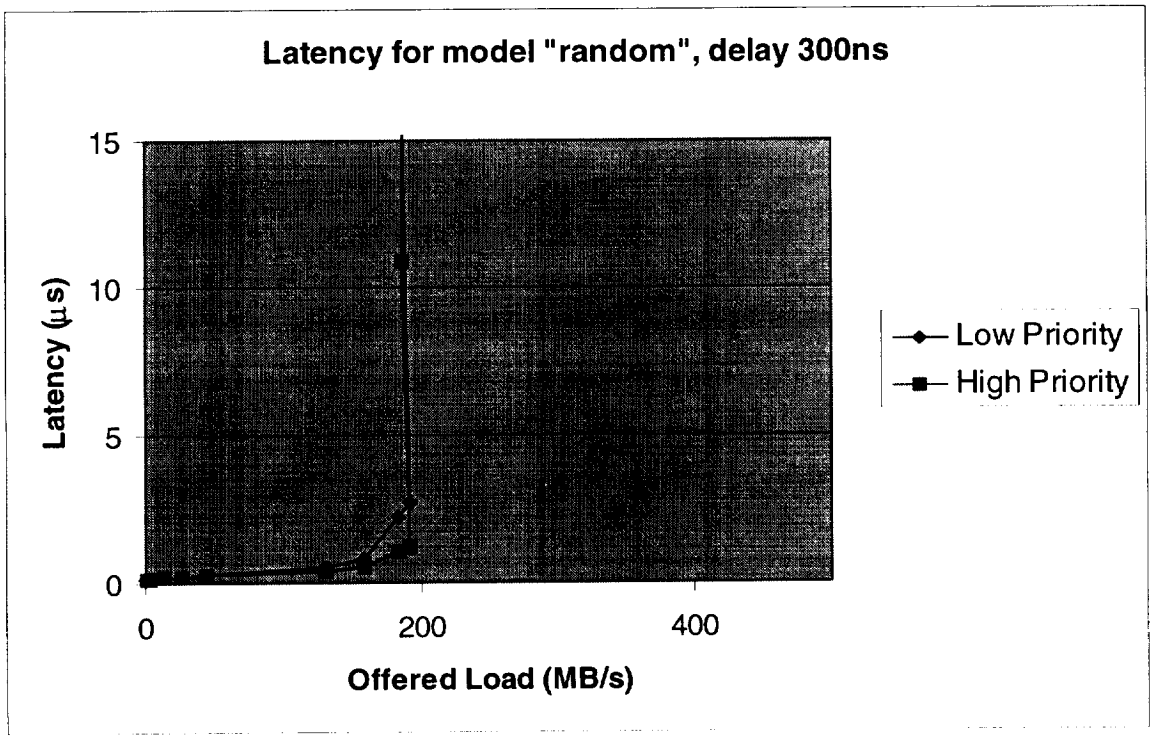

Figure 3: Latencies of low and high priority packets vs. the offered load for a system with delay 300ns

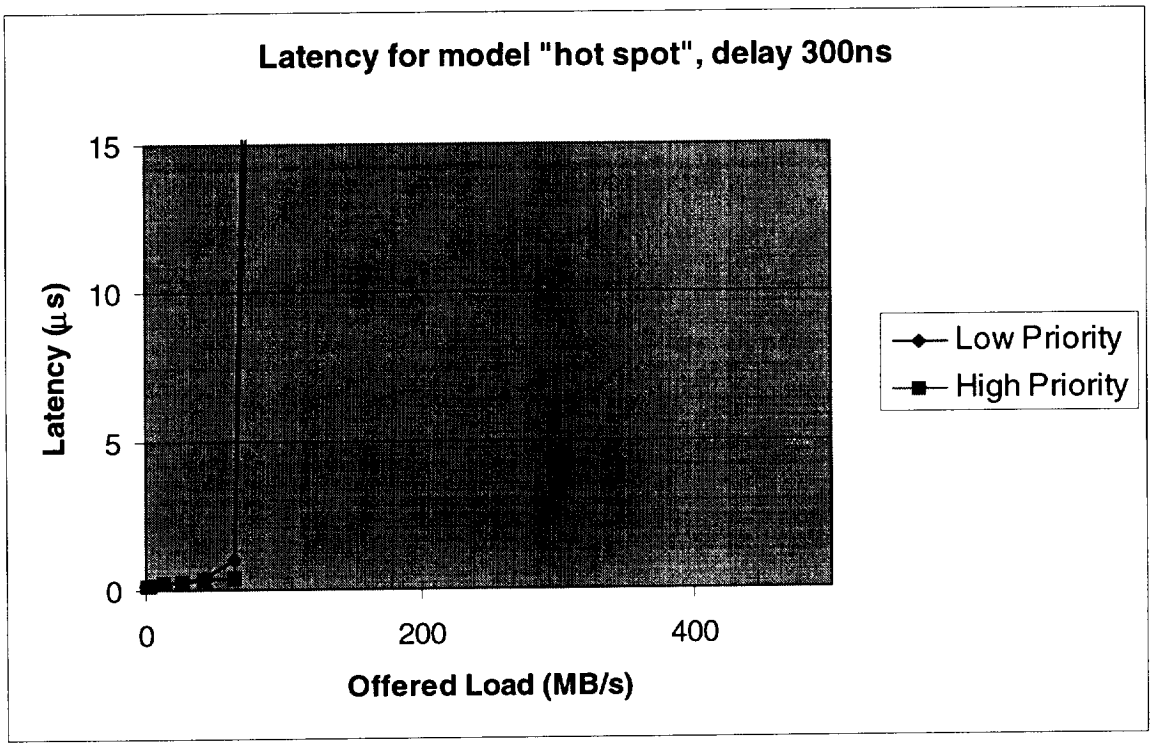

Figure 4: Latencies of low and high priority packets vs. the offered load for a system with "hot spot and delay 3000ns

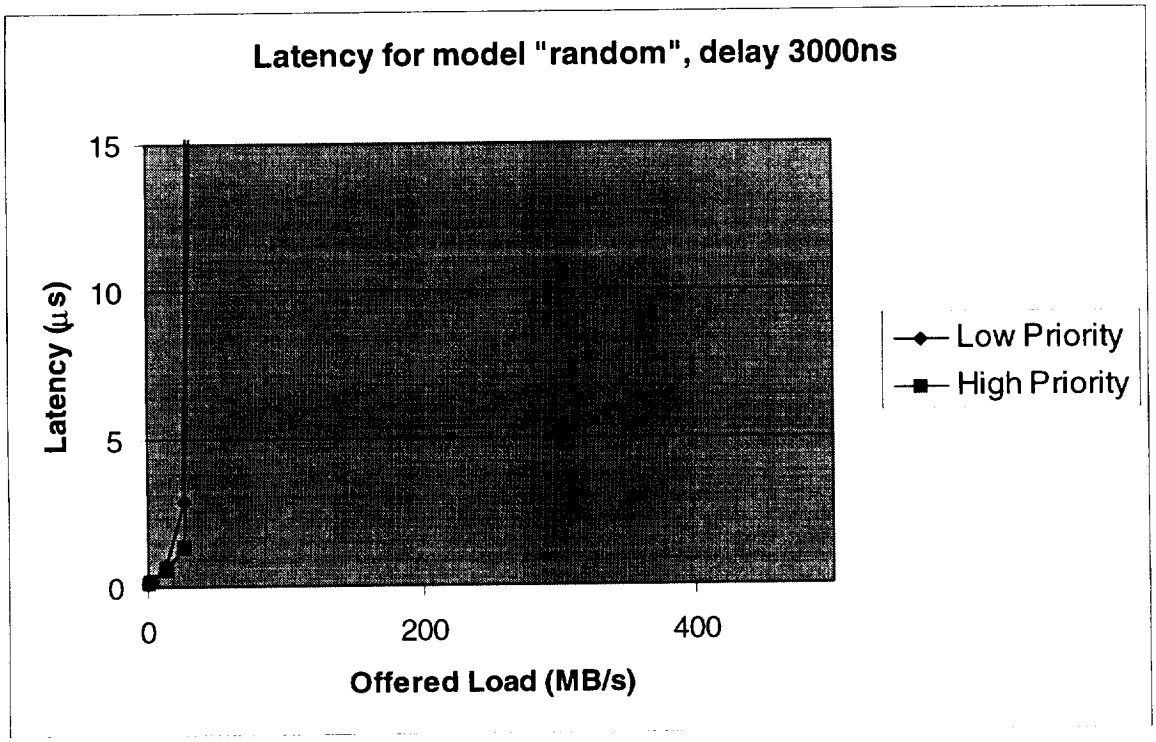

Figure 5: Latency of high and low priority vs. offered load with delay 3000ns

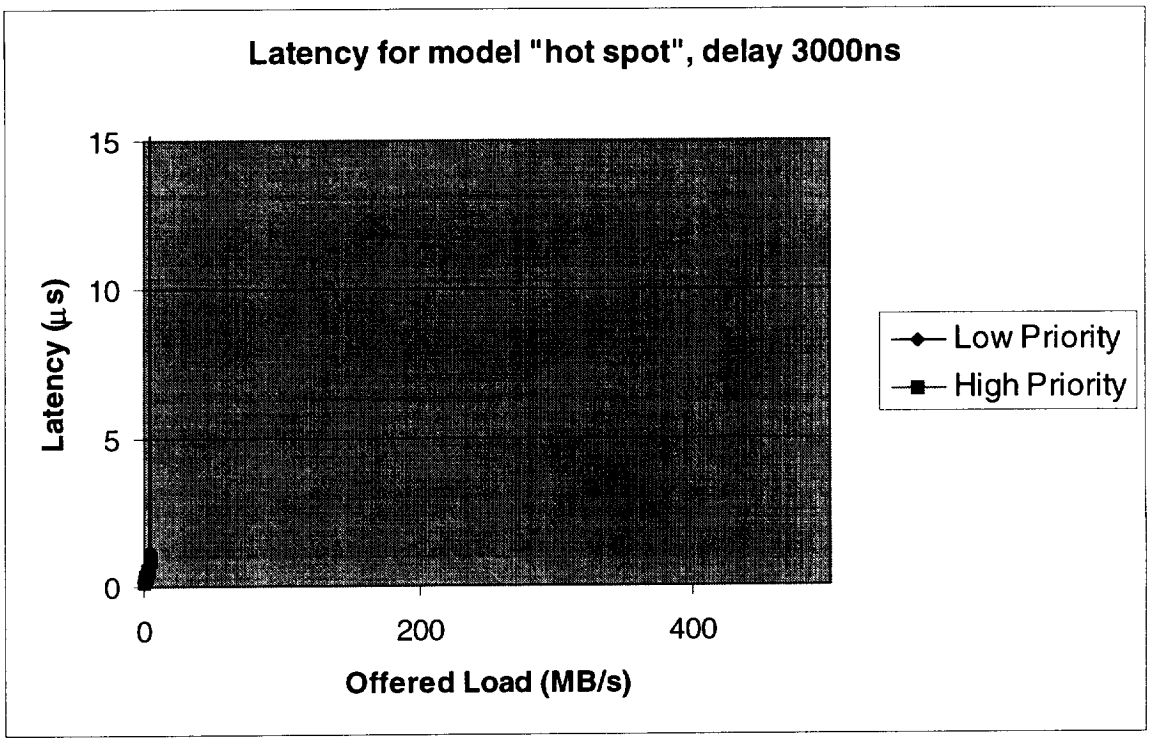

Figure 6: Latencies of low and high priority packets vs. the offered load for a system with "hot spot " and delay 3000ns

Figure 7 through Figure 9 show the cases where bypass control symbols are turned off but the control of input queues are still in effect. In those cases, the system slightly favors the high priority traffic, but not statistically significant.

To examine the effect of the bypass control, Figure 7 and Figure 1, Figure 8 and Figure 3, and Figure 9 and Figure 5, are compared. The bypass stop control symbol is generated whenever the node has to transmit a new packet of higher priority than the packet in its bypass. It is circulated to all nodes around the ring telling them to stop sending low priority packets through the bypass of the node that sent it. Once the output queues of this node are cleared of all higher priority packets (ones that met all the priority restrictions between the source and destination nodes) the start bypass control symbol is generated. There can be a large number of bypass control symbols generated depending on loading and ratio of low to high priority packets generated. We studied the impact of these control symbols on the system performance by turning off their generation and comparing the new set of results obtained with our previous results. We first measured average latencies for the random model with various input request processing delays (Figures 7 through 9). The graphs follow a similar trend in that there is no appreciable difference in the average latencies of the high and the low priority packets at low offered loads. The difference is visible at higher loads where the effects of prioritized input queues and input queue control symbols kicks in once they start filling up and high priority packets have lower latencies. We compared the average latencies for the same request processing delay for the case when all types of

control symbols (including bypass) were generated to that when generation of bypass control symbols was turned off. In general (see Figures 1 and 7, and 3 and 8) we found that the model with NO bypass control symbols generated could sustain lower latencies for higher offered loads, while at lower loads they performed equally well.

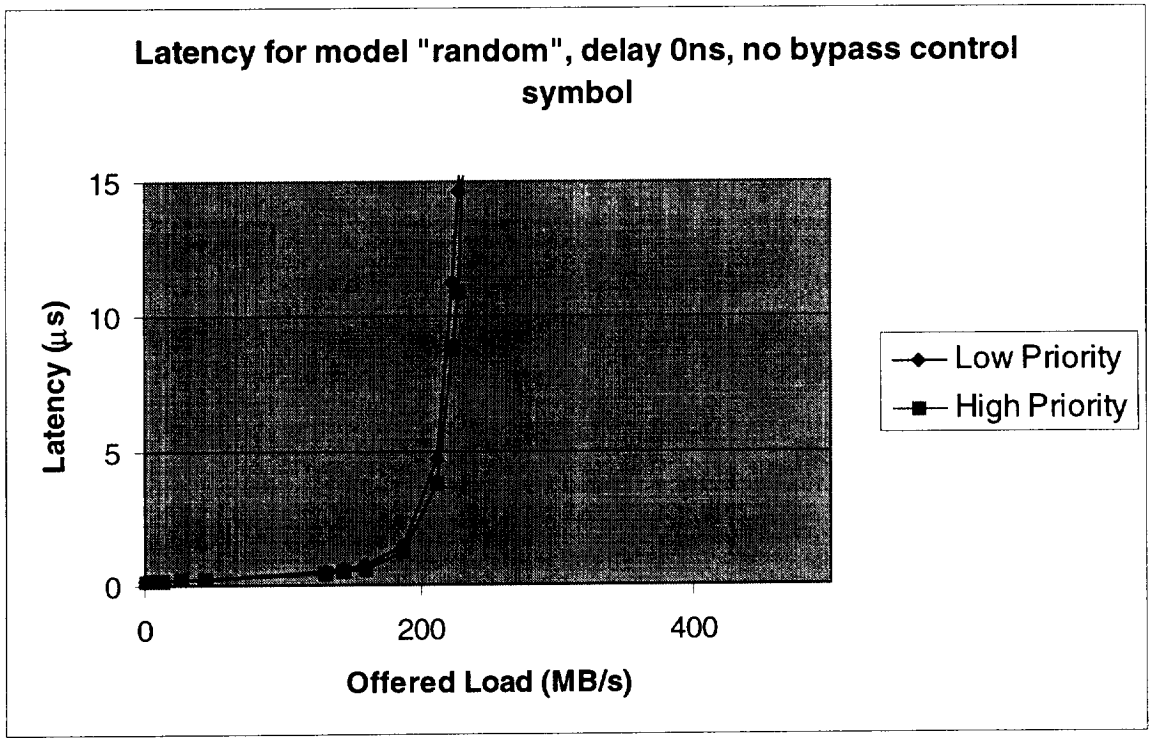

Figure 7: Latency of high and low priority vs. offered load with delay 0ns and no bypass control symbols

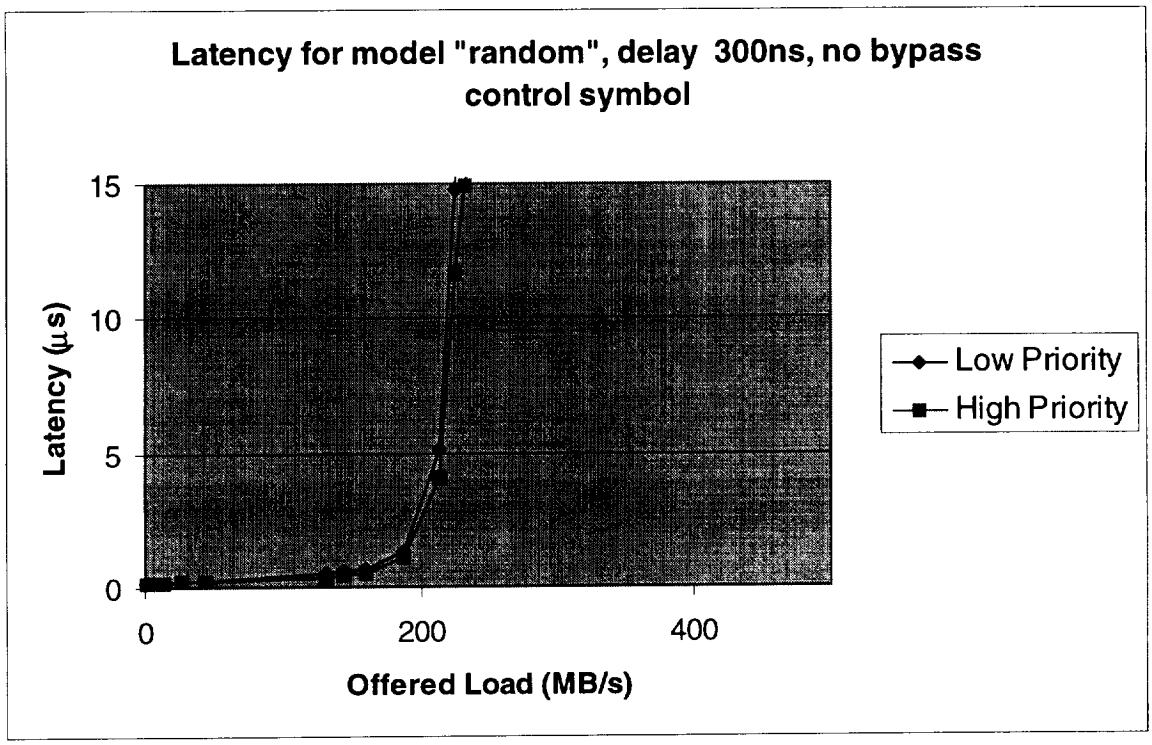

Figure 8: Latencies of low and high priority packets vs. the offered load for delay 300ns

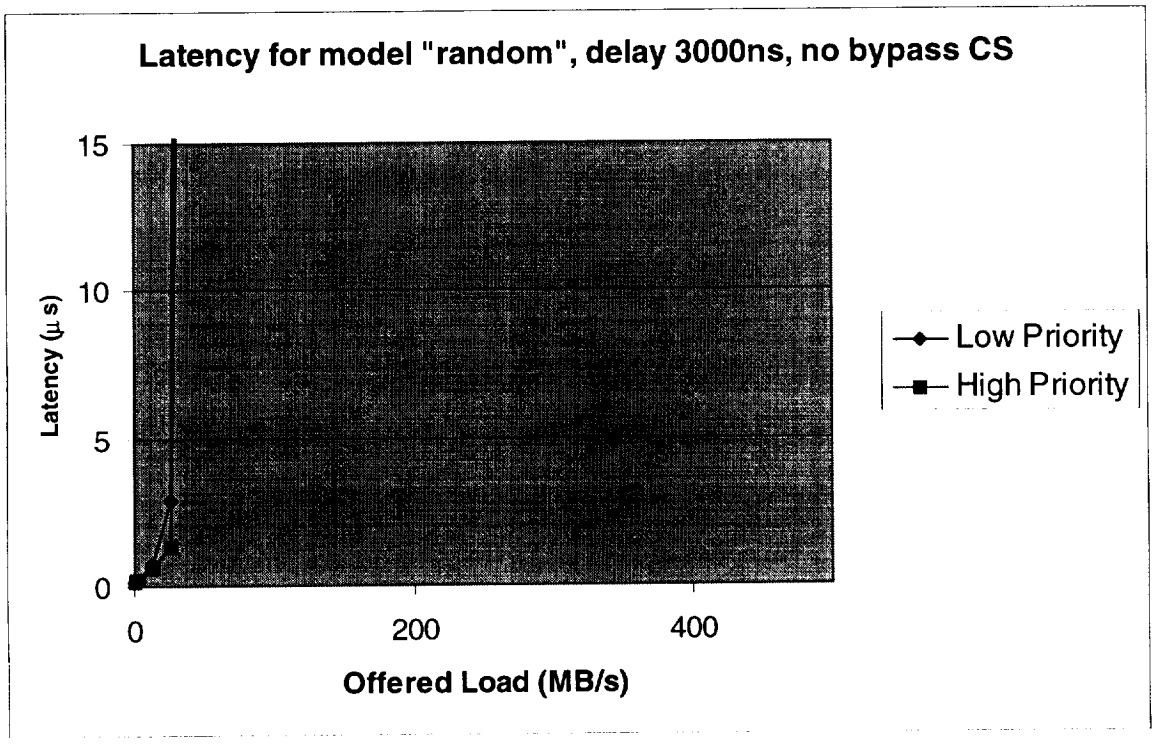

Figure 9: Latency of high and low priority vs. offered load with delay 3000ns and no bypass control symbols

For some real-time applications it is necessary to have an upper bound on the latency. For this reason we measured the maximum latencies for the various processing delays in both types of models (Figures 10 through 18). In general it can be seen that all graphs follow a similar trend in that the high priority packets have a lower maximum latency than low priority packets. On comparing maximum latencies (Figures 10 and 16, and 12 and 17) it is clear that the bypass control symbols are successful in lowering the maximum latency of high priority packets. Considering the early results that bypass control did not reduce the average latency, the result here indicates that it gives the high priority packets a more uniform latency.

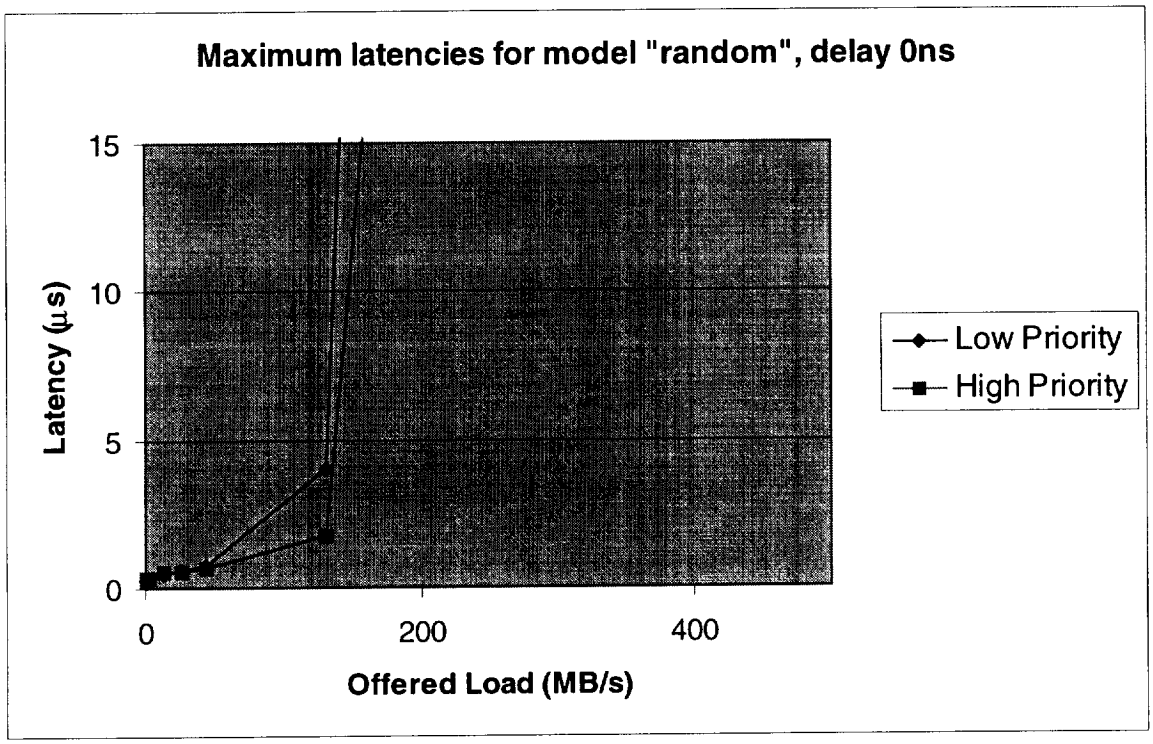

Figure 10: Maximum latencies vs. offered load with delay 0ns

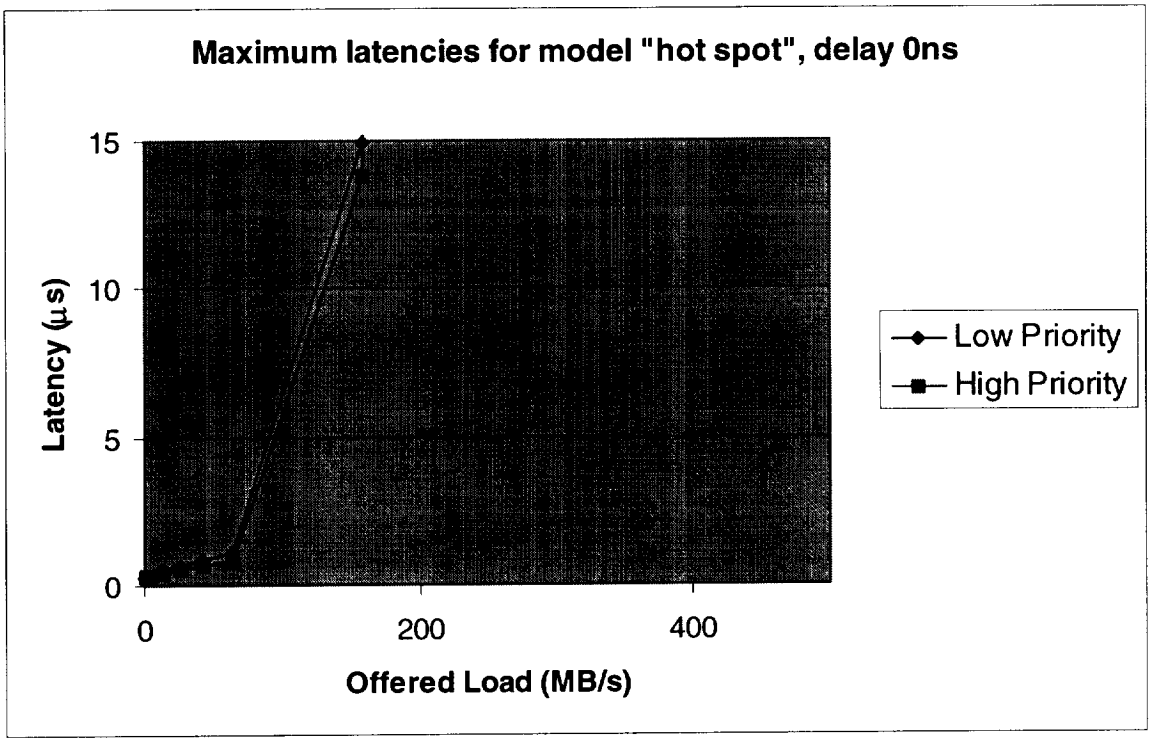

Figure 11: Maximum latencies vs. offered load with delay 0ns for "hot spot" model

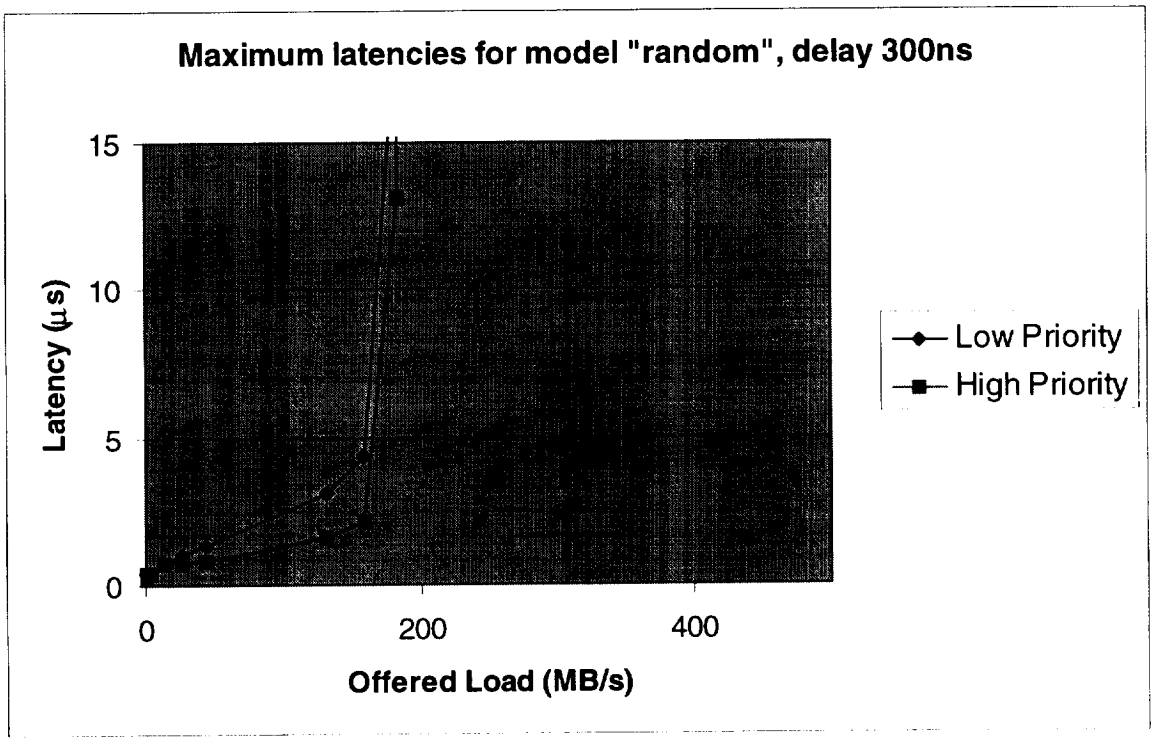

Figure 12: Maximum latencies vs. offered load with delay 300ns

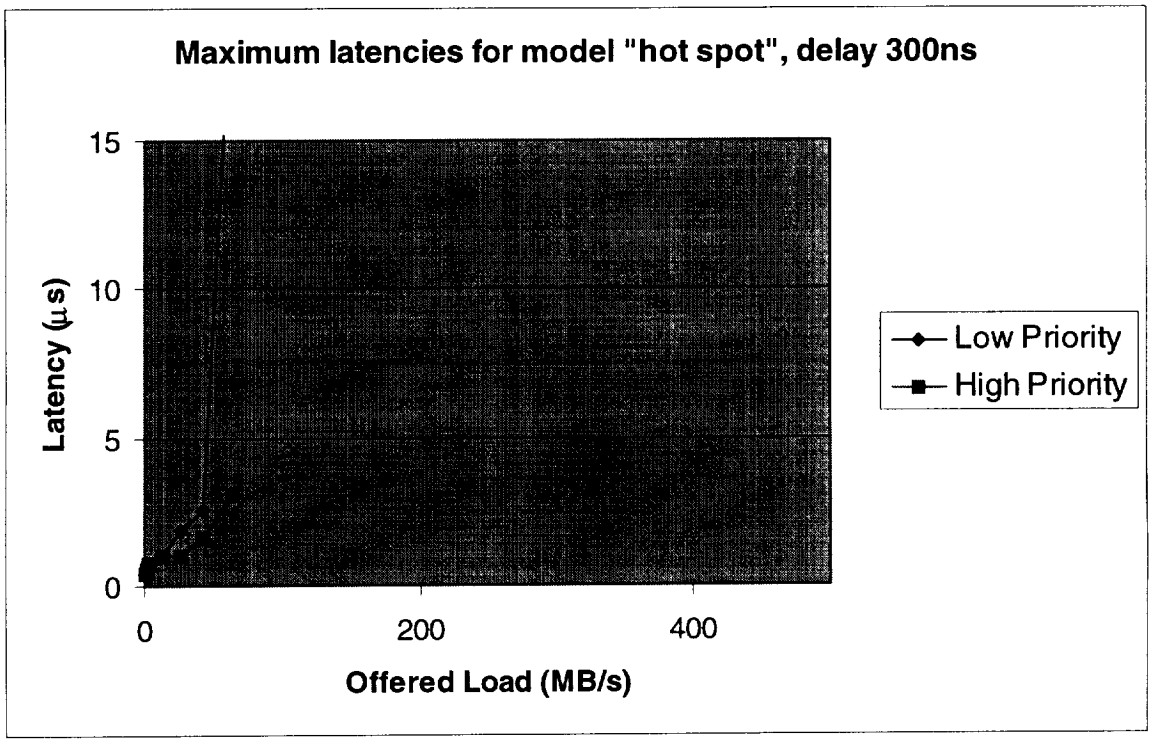

Figure 13: Maximum latencies vs. offered load with delay 300ns for "hot spot" model

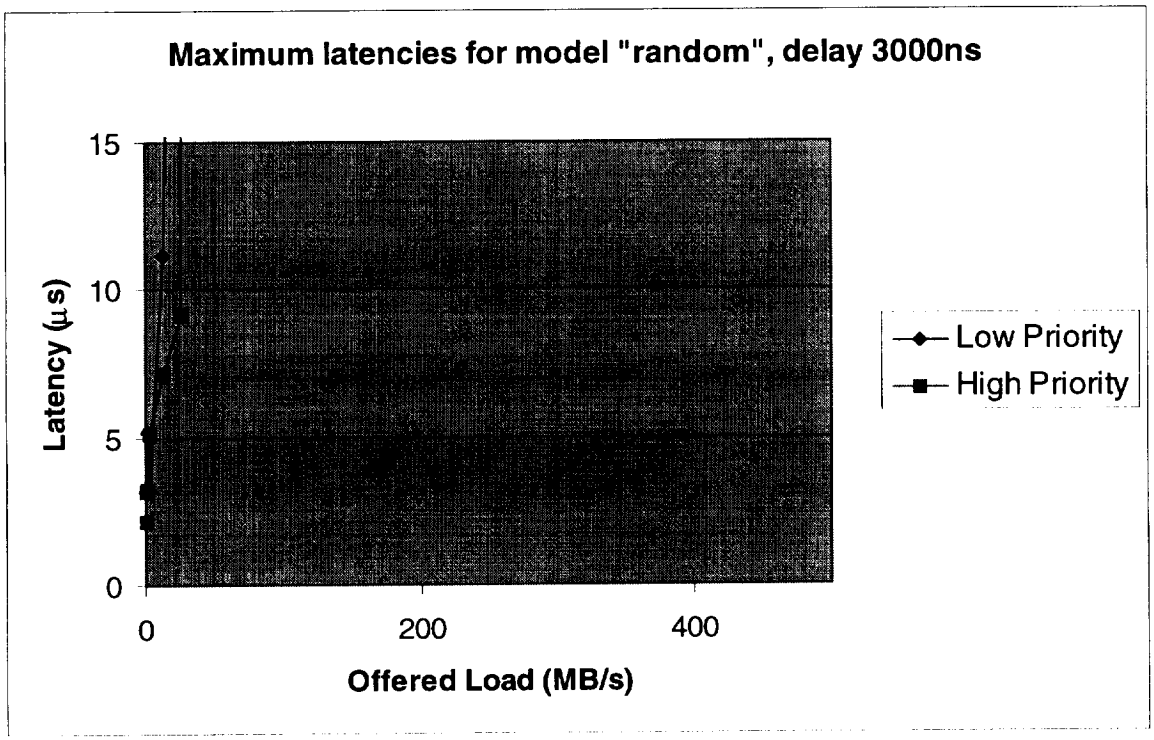

Figure 14: Maximum latencies vs. offered load with delay 3000ns

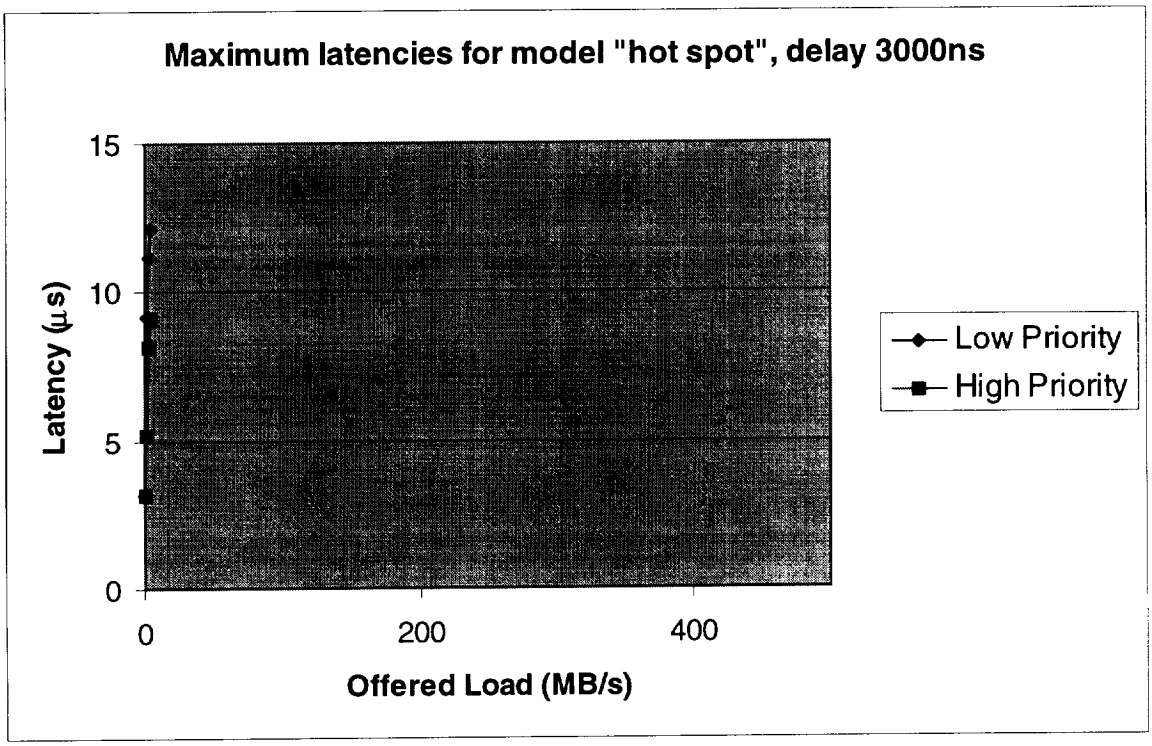

Figure 15: Maximum latencies vs. offered load with delay 3000ns for "hot spot" model

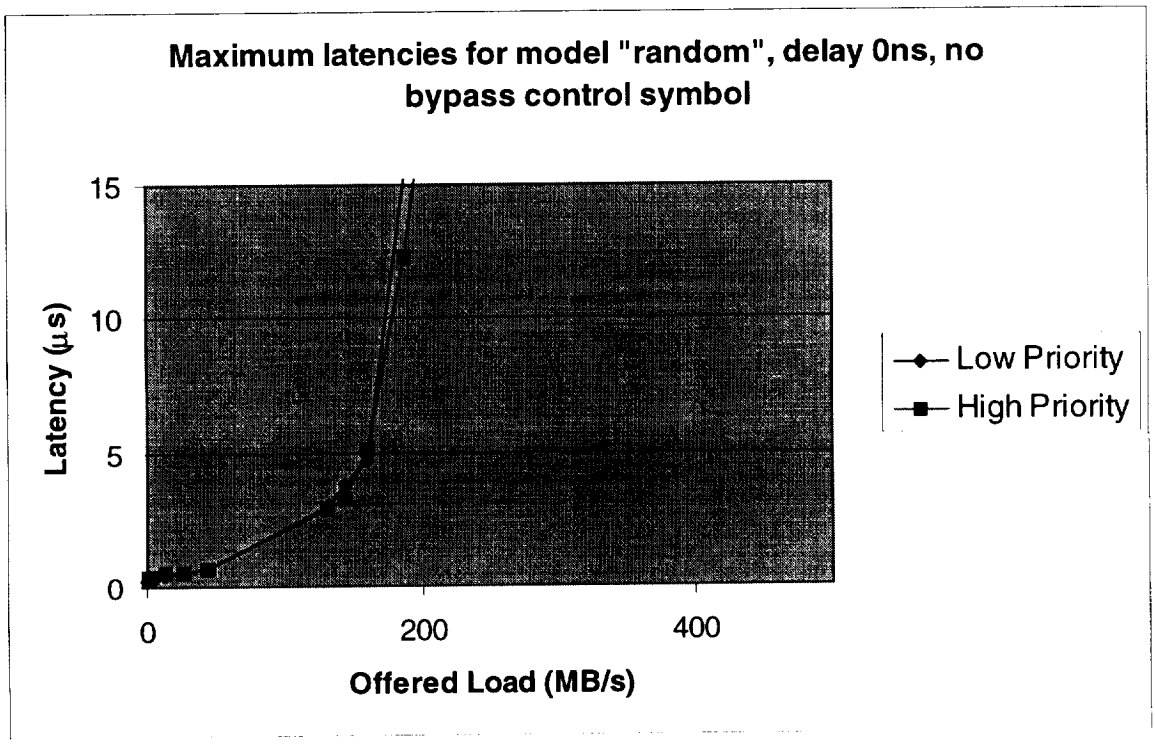

Figure 16: Maximum latencies vs. offered load with delay 0ns and no bypass control symbol

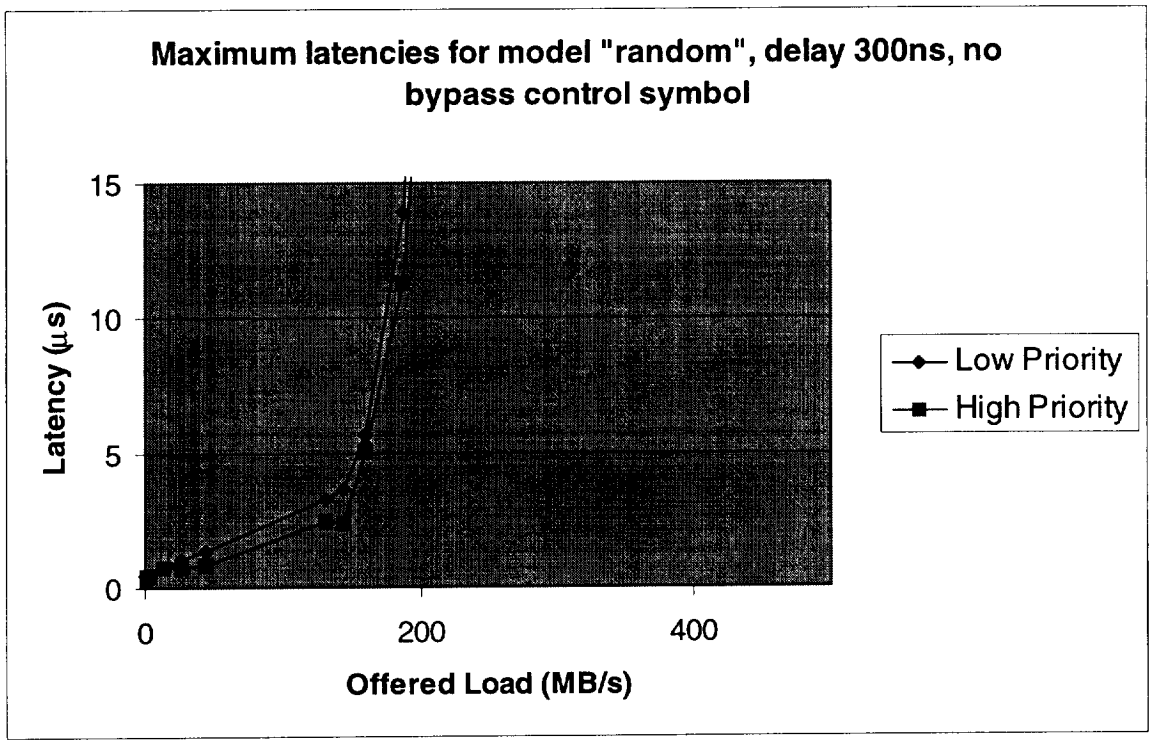

Figure 17: Maximum latencies vs. offered load with delay 300ns and no bypass control symbol

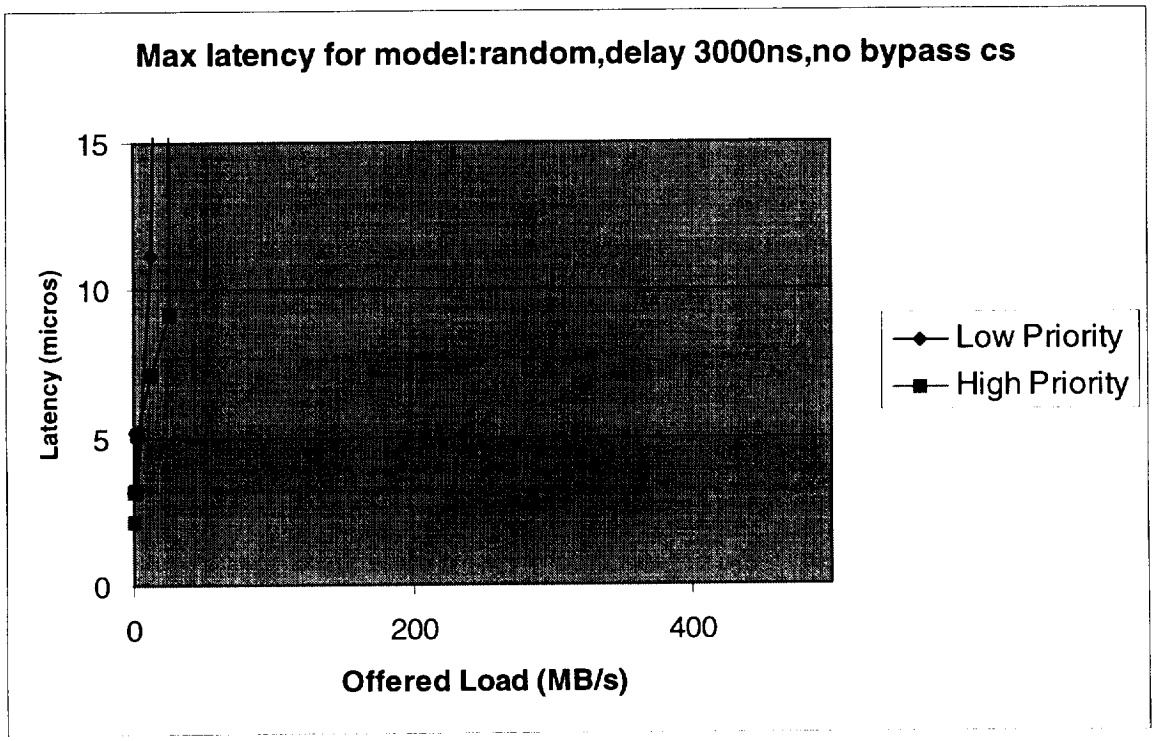

Figure 18: Maximum latencies vs. offered load with delay 3000ns and no bypass control symbol

4.2 The plots of offered load vs. throughput follow:

Figures 19 through 27 show the throughput achieved as offered loads change in all of the above cases. The throughput is measured as the net data traffic. When a memory request is made, a request of 16 bytes is sent to the target node, and a responding packet containing 64 bytes of data is sent back to the requester. Only the net 64 bytes of data are counted towards the throughput. The offered loads shown in the charts are the traffic of requesting packets. Therefore there is a 4:1 ratio between the throughput and the offered load, i.e., every 16 bytes of request traffic causes 64 bytes net data traffic.

The graphs clearly show that the system saturates earlier with the hot spot. An increase in the number of busy retries and shutting down the system with control symbols contribute to the above.

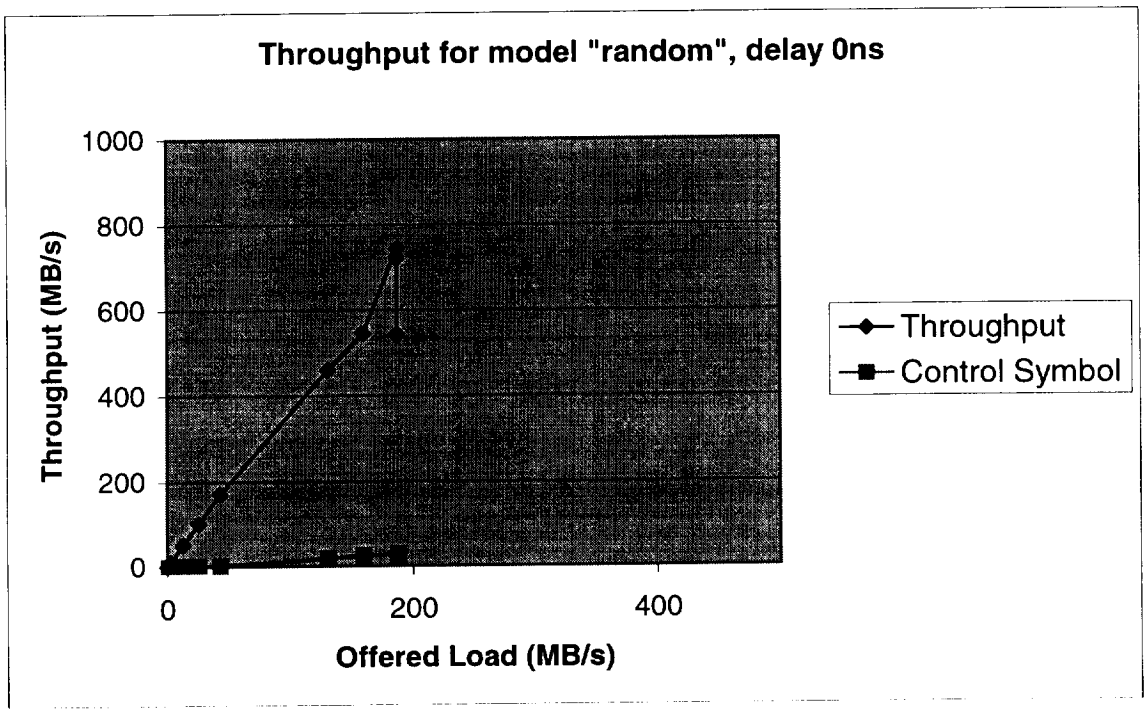

Figure 19: Offered Load vs. overall system throughput with delay 0ns. Bandwidth consumed by control symbols is included

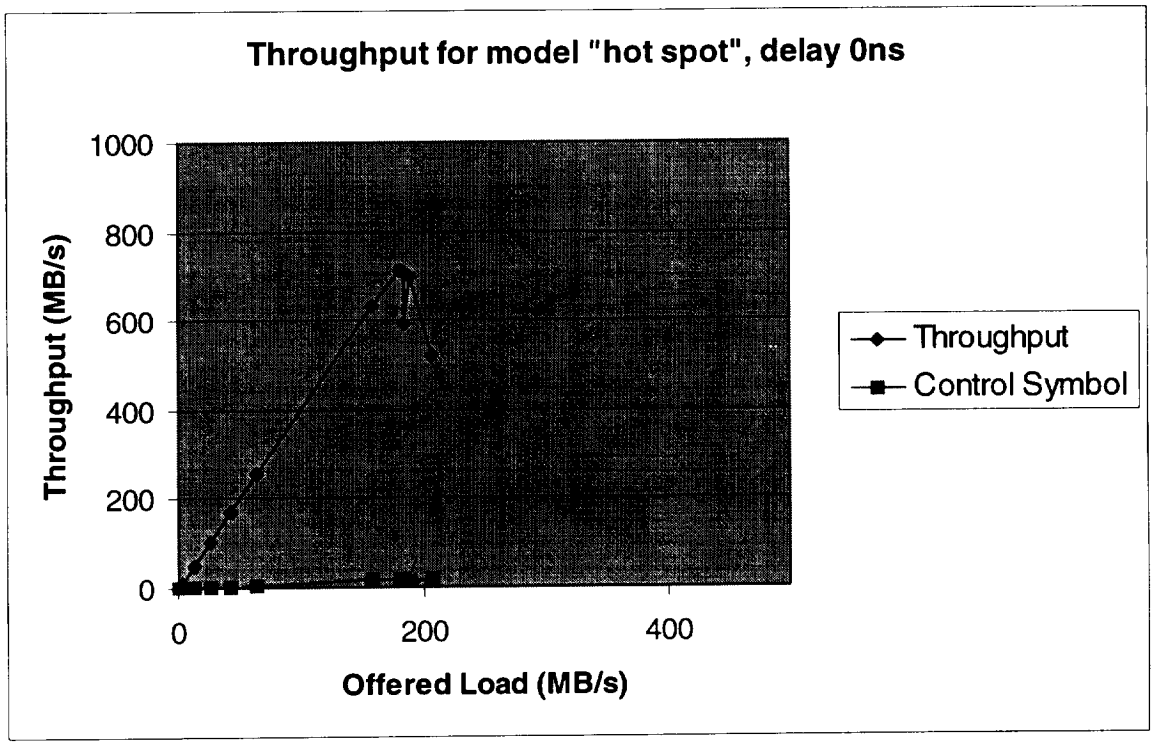

Figure 20: Offered Load vs. overall system throughput in a "hot spot" model with delay 0ns. Bandwidth consumed by control symbols is included.

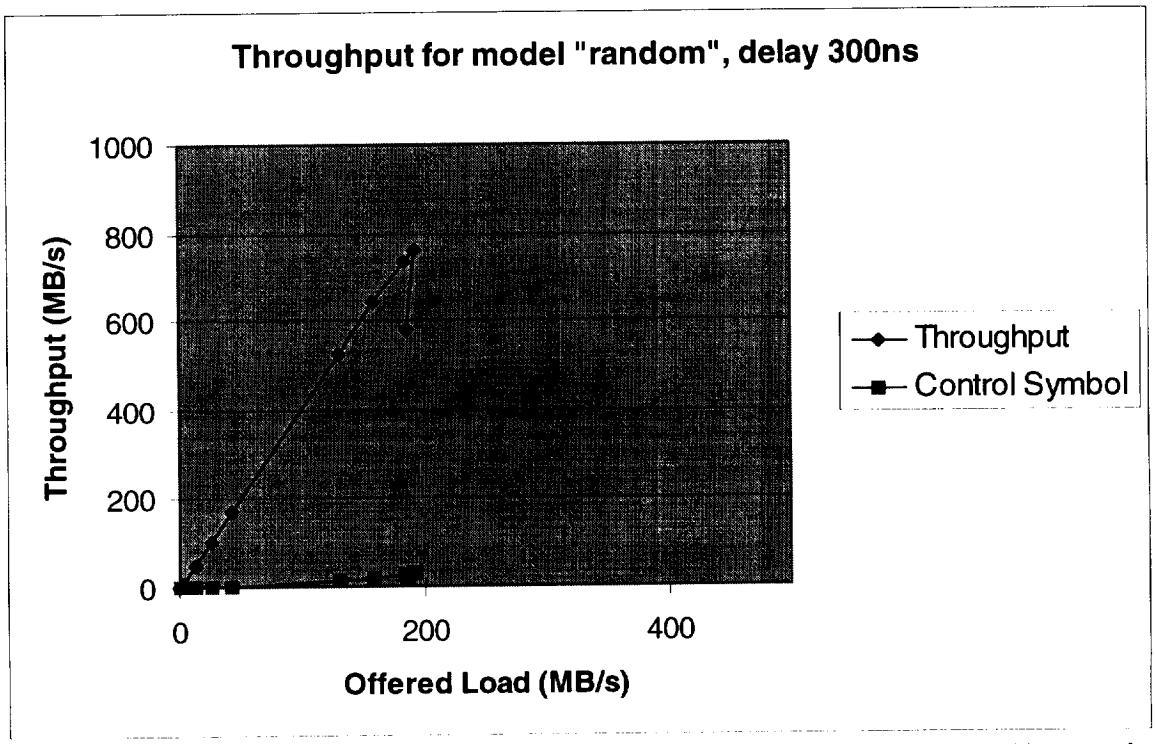

Figure 21: Offered Load vs. overall system throughput with delay 300ns. Bandwidth consumed by control symbols is .included

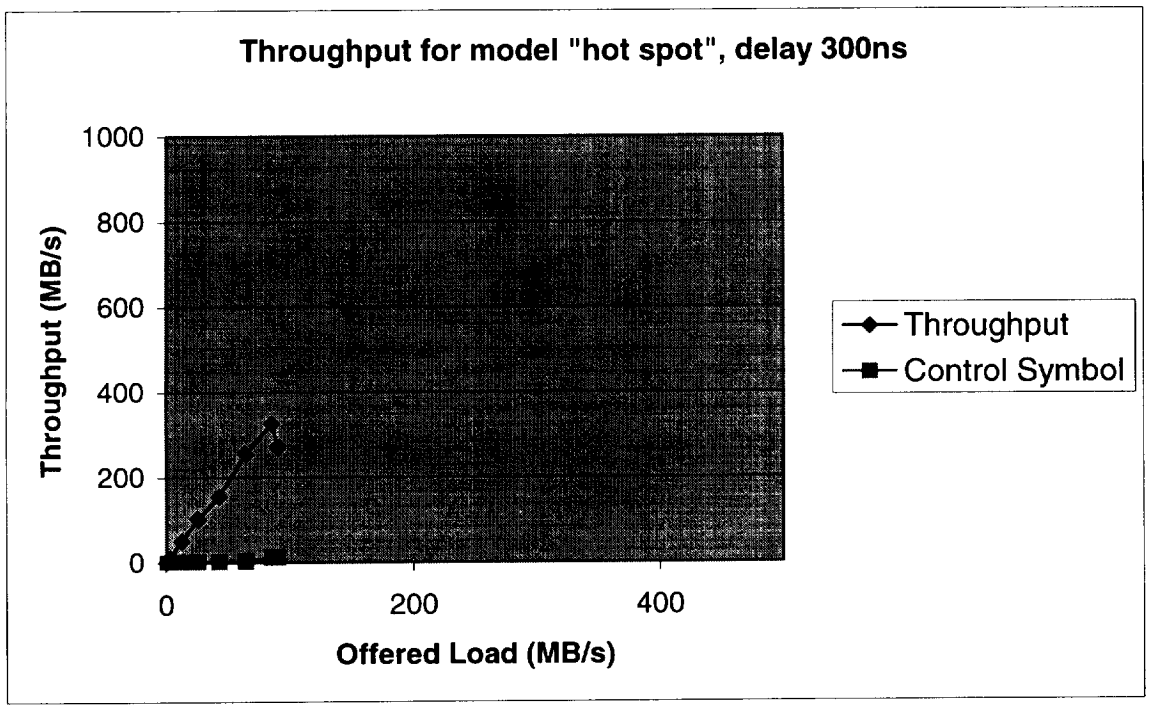

Figure 22: Offered Load vs. overall system throughput in a "hot spot" model with delay 300ns. Bandwidth consumed by control symbol is included.

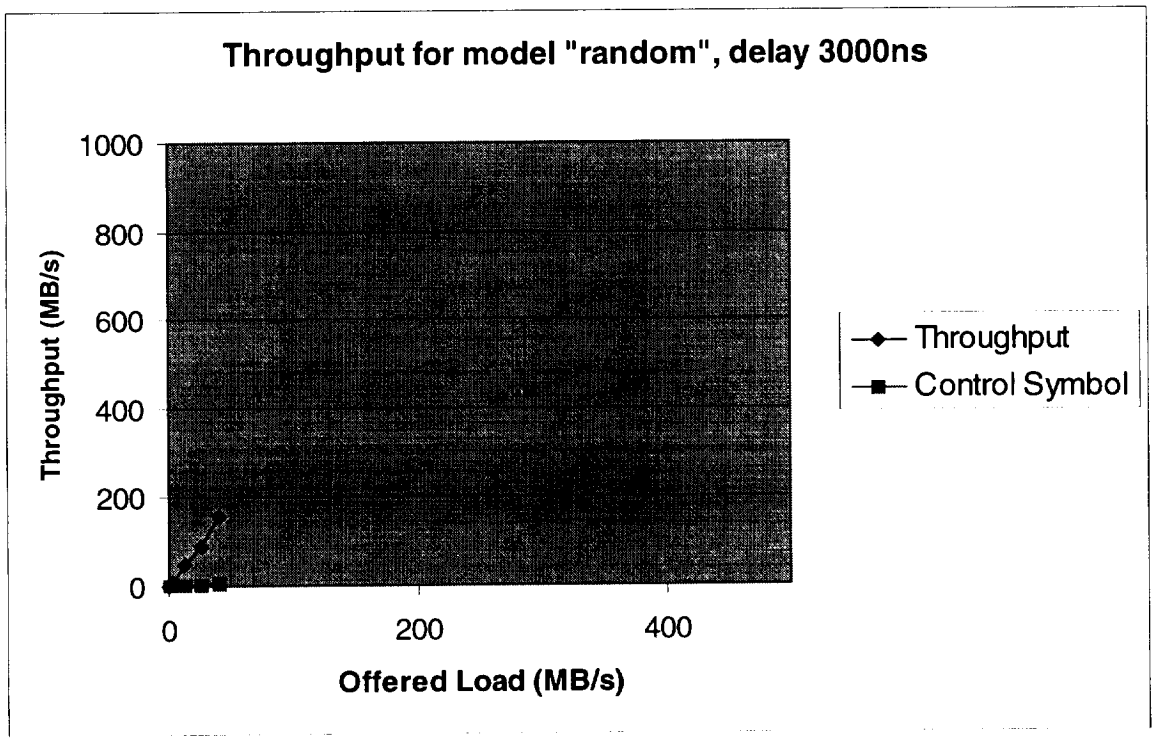

Figure 23: Offered Load vs. overall system throughput with delay 3000ns. Bandwidth consumed by control symbols is .included

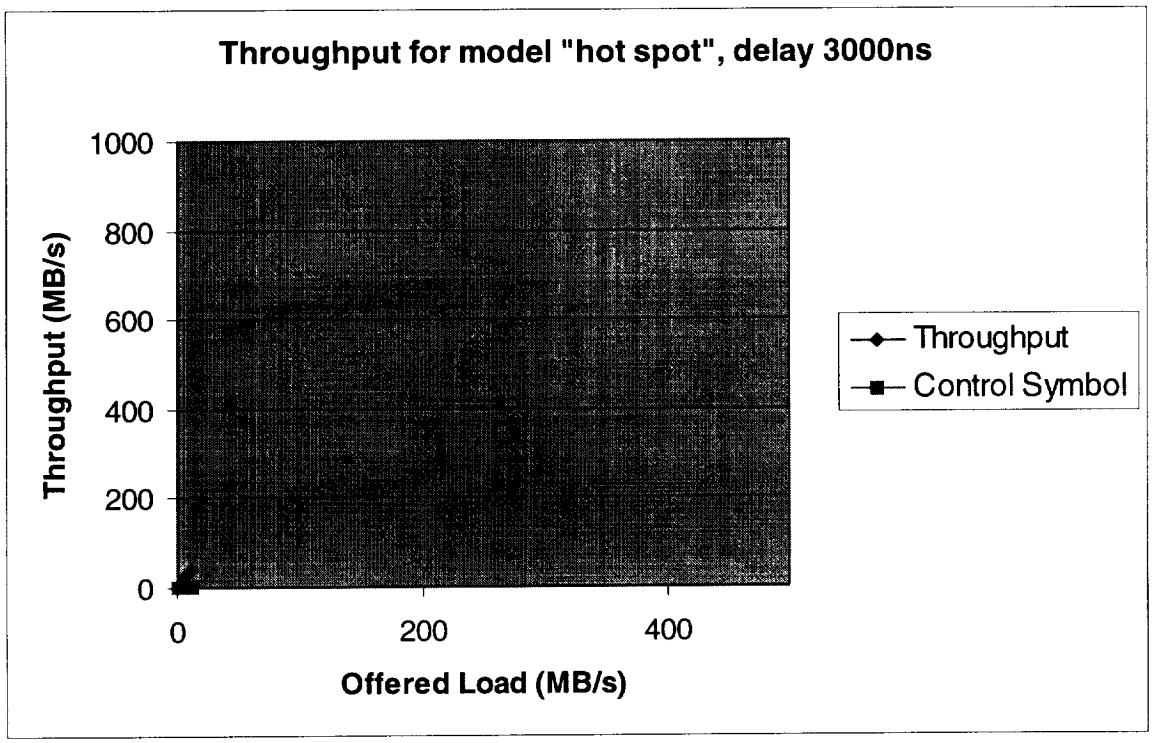

Figure 24: Offered Load vs. overall system throughput in a "hot spot" model with delay 3000ns. Bandwidth consumed by control symbols is included.

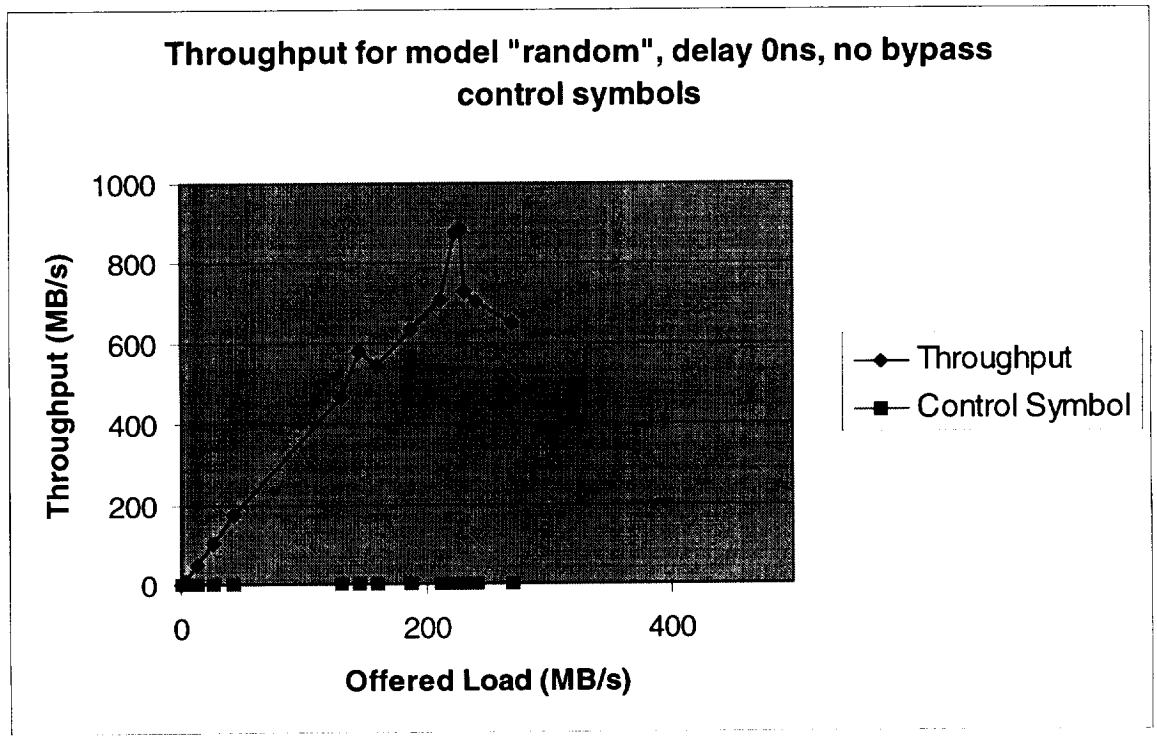

Figure 25: Offered Load vs. overall system throughput with delay 0ns. Bandwidth consumed by input queue control included

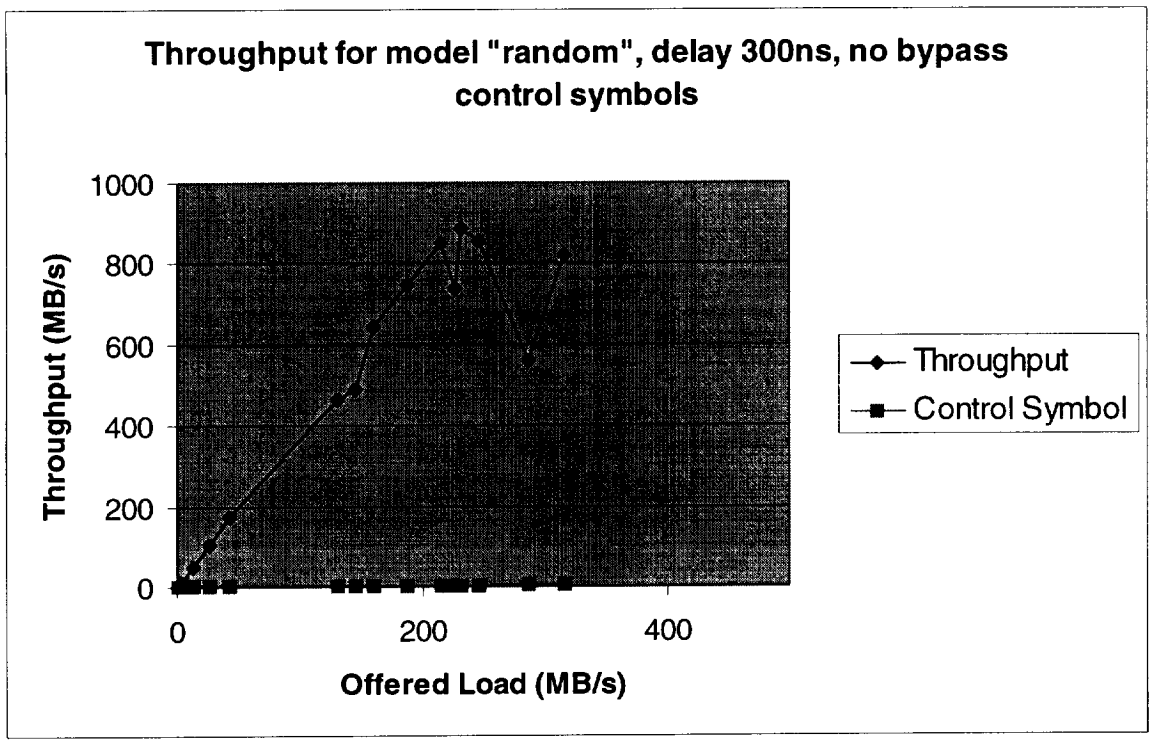

Figure 26: Offered Load vs. overall system throughput with delay 300ns. Bandwidth consumed by input queue control .included

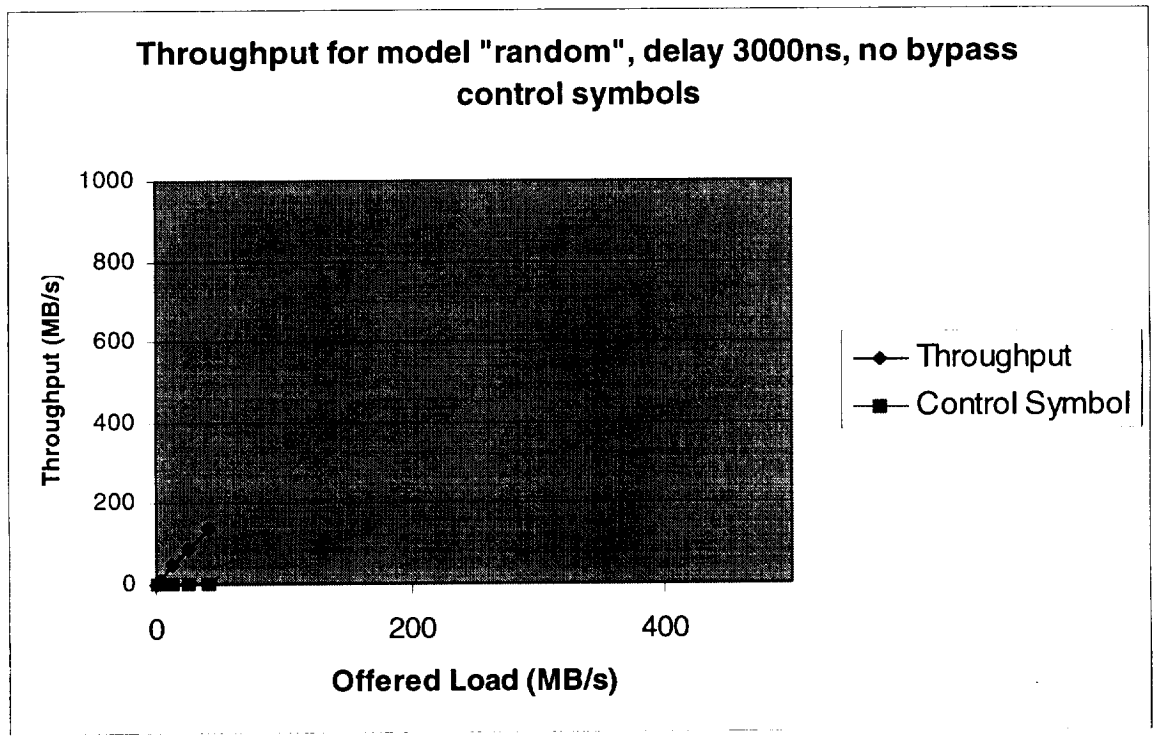

Figure 27: Offered Load vs. overall system throughput with delay 3000ns. Bandwidth consumed by input queue contr .included

In comparing the graphs which show the variation of the throughput with the offered load, we find that the initial parts of the graphs are similar, but eventually higher throughputs are obtained for the same offered loads where service times were lower. This is very much in keeping with our intuition.

5 Conclusions

We have investigated the control symbol based priority protocol for SCI-based systems. While the protocol results in some improvement to the latency of the high priority packet, the improvement is not statistically significant. On comparing maxmum latencies (Figure 10 and 16, and 12 and 17) it is clear that the bypass control symbols are successful in lowering the maximum latency of the high priority packets but they also cause saturation to occur at lower offered load.

6 Future work

In this simulation the HIGH and LOW watermark levels were set at 4 and 1 respectively. It would be interesting to study the effect of varying these on the latencies of packets and the number of control symbols generated and the bandwidth consumed by them. This study used a single ring with a fixed number of nodes. Multiple interconnected ring topologies could also be simulated which would offer a more realistic view.

The enhancement to the protocol is currently being studied. We observed that the bypass control symbol may have shut down the traffic unnecessarily. An extended bypass FIFO queue may allow us to inject high priority packets into the ring without sending the bypass control symbols.

Acknowledgement

We gratefully acknowledge the support of our sponsors Mr. Ralph Lachenmaier and Tom Stretch, for their guidance in this project as well as their financial support.

References

[RTSCI1] A Draft Proposal for a SCI/Real-Time Protocol using Directed Flow Control Symbols Ralph Lachenmaier, NAWC/AD Warminster and Tom Stretch, AMPAC, Inc

[SCI1] Scalable Coherent Interface, ANSI/IEEE Standard 1596-1992, IEEE Service Center, Piscataway, New Jersey, 1993.

[RTSCI2] Isaac Chu, Robert Grygiel and Ronald D. Fellman, "An Open-Loop Flow-Control Protocol for SCI-RT", *Proceedings of the 2nd International Workshop on SCI-based High-Performance Low-Cost Computing, March 26,1996.*

[SCI2] LSI Logic, "Interfacing to SCI", February 1995.

APPENDIX

Description of SES/Workbench

SES/workbench is an integrated collection of software tools for specifying and evaluating system design. It includes SES/design, SES/tran, SES/sim, SES/scope and SES/report. SES design is a design interface (or design capture) module for specifying a system design. . SES tran is a translation module that converts graphical models created under SES/design into SES/sim. The SES/sim is a translation and simulation module for converting a design specification into an executable module. The executable can be run providing simulation and statistical output. The SES/sim language is a superset of C. SES/scope is an animation module that provides the ability to observe and debug an existing simulation module.

Description of model

The SES node and links are modeled using SES Workbench. **Service nodes** "*input_request_q*" and "*input_response_q*" model the input queues. The **user node** "*address_decode*" routes the incoming packets correctly. The **blocking node** "*bypass_fifo*" models the bypass. The **user**, **block** and **interrupt nodes** collectively model the creation and sending of a stop control symbol for the input queue (labeled "*send_stop_symbol*") and place it on the link when available. The "*send_start_symbol* " and "*send_start_symbol1*" generate start control symbol for the input response and request queues respectively. Similarly "*send_stop_byp_symbol*" and "*send_start_byp_symbol*" create bypass control symbols.

The **user node** and **interrupt node** "*create_response* " and "*route2to_output*_q" create a response packet to the request after the delay specified by delay node "*process_delay*". This packet is routed to the "*output_response_q*". Access to the link is controlled by a semaphore "*get_token*" which ensures that no packet can "cut in-between" when a packet is being transmitted on the link.

The **declaration node** "*real_time*" contains the DIQS, Restricted Traffic and Busied Traffic register data structures.

The **blocking node** "*send_echo*" and the **interrupt node** "*route_to_echo_q*" create an echo when the last symbol in a packet is received (Note: this could be modified for the last but one). The echo is routed to the echo_q.

The output queues are "*output_response_q*" and "*output_request_q*". If a packet cannot be

transmitted immediately from these it is stored in the **blocking node** "*wait_for_clearance*". In a real system this could be the output queue itself. The logic implemented by these queues is simply to evaluate all the packets waiting in the output queue and transmit those that meet all the priority restrictions between source and destination. The queue in this node is polled every time a control symbol arrives to see if any of packets meet the priority requirement after the DIQS/ Restricted traffic registers have been updated.